МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №1

Специальность ИИ24

Выполнил

Д. С. Сухаревич,

студент группы ИИ24

Проверила

К. В. Андренко,

Преподаватель-стажер кафедры ИИТ,

« » 2025 г.

Брест 2025

**Цель работы:** научиться конструировать нейросетевые классификаторы и выполнять их обучение на известных выборках компьютерного зрения

Задание 1. Выполнить конструирование своей модели СНС, обучить ее на выборке по заданию (использовать torchvision.datasets). Предпочтение отдавать как можно более простым архитектурам, базирующимся на базовых типах слоев (сверточный, полносвязный, подвыборочный, слой нелинейного преобразования). Оценить эффективность обучения на тестовой выборке, построить график изменения ошибки (matplotlib);

**Выполнение:**

# Код программы

**import** argparse  
**import** os  
**import** time  
**from** PIL **import** Image  
**import** numpy **as** np  
**import** torch  
**import** torch.nn **as** nn  
**import** torch.optim **as** optim  
**import** torchvision  
**import** torchvision.transforms **as** transforms  
**import** matplotlib.pyplot **as** pltparser = argparse.ArgumentParser()  
parser.add\_argument(**'--epochs'**, type=int, default=30)  
parser.add\_argument(**'--batch-size'**, type=int, default=128)  
parser.add\_argument(**'--lr'**, type=float, default=1e-3)parser.add\_argument(**'--momentum'**, type=float, default=0.9)parser.add\_argument(**'--weight-decay'**, type=float, default=1e-4)  
parser.add\_argument(**'--save-dir'**, type=str, default=**'checkpoints'**)  
parser.add\_argument(**'--use-cuda'**, action=**'store\_true'**)  
parser.add\_argument(**'--visualize-image'**, type=str, default=**None**, help=**'path to image to run through network (optional)'**)  
args = parser.parse\_args()  
device = torch.device(**'cuda' if** args.use\_cuda **and** torch.cuda.is\_available() **else 'cpu'**)  
os.makedirs(args.save\_dir, exist\_ok=**True**)**class** SimpleCNN(nn.Module):  
 **def** \_\_init\_\_(self, num\_classes=10):  
 super().\_\_init\_\_()self.features = nn.Sequential(  
 nn.Conv2d(3, 32, kernel\_size=3, padding=1), *# 32x32x32* nn.ReLU(inplace=**True**),  
 nn.MaxPool2d(2, 2), *# 32x16x16* nn.Conv2d(32, 64, kernel\_size=3, padding=1), *# 64x16x16* nn.ReLU(inplace=**True**),  
 nn.MaxPool2d(2, 2), *# 64x8x8* nn.Conv2d(64, 128, kernel\_size=3, padding=1),*# 128x8x8* nn.ReLU(inplace=**True**),  
 nn.MaxPool2d(2, 2), *# 128x4x4* )  
 self.classifier = nn.Sequential(  
 nn.Flatten(),  
 nn.Linear(128\*4\*4, 256),  
 nn.ReLU(inplace=**True**),  
 nn.Dropout(0.5),  
 nn.Linear(256, num\_classes),  
 )  
 **def** forward(self, x):  
 x = self.features(x)  
 x = self.classifier(x)  
 **return** xtransform\_train = transforms.Compose([  
 transforms.RandomHorizontalFlip(),  
 transforms.RandomCrop(32, padding=4),  
 transforms.ToTensor(),  
 transforms.Normalize((0.4914, 0.4822, 0.4465),  
 (0.2023, 0.1994, 0.2010)),  
])  
transform\_test = transforms.Compose([  
 transforms.ToTensor(),  
 transforms.Normalize((0.4914, 0.4822, 0.4465),  
 (0.2023, 0.1994, 0.2010)),  
])  
trainset = torchvision.datasets.CIFAR10(root=**'./data'**, train=**True**, download=**True**, transform=transform\_train)  
trainloader = torch.utils.data.DataLoader(trainset, batch\_size=args.batch\_size, shuffle=**True**, num\_workers=4)  
testset = torchvision.datasets.CIFAR10(root=**'./data'**, train=**False**, download=**True**, transform=transform\_test)  
testloader = torch.utils.data.DataLoader(testset, batch\_size=100, shuffle=**False**, num\_workers=4)  
classes = trainset.classesmodel = SimpleCNN(num\_classes=10).to(device)  
criterion = nn.CrossEntropyLoss()  
optimizer = optim.RMSprop(model.parameters(), lr=args.lr, weight\_decay=args.weight\_decay)scheduler = optim.lr\_scheduler.StepLR(optimizer, step\_size=15, gamma=0.1)**def** evaluate(loader):  
 model.eval()  
 correct = 0  
 total = 0  
 running\_loss = 0.0  
 **with** torch.no\_grad():  
 **for** inputs, targets **in** loader:  
 inputs, targets = inputs.to(device), targets.to(device)  
 outputs = model(inputs)  
 loss = criterion(outputs, targets)  
 running\_loss += loss.item() \* inputs.size(0)  
 \_, predicted = outputs.max(1)  
 total += targets.size(0)  
 correct += predicted.eq(targets).sum().item()  
 **return** running\_loss / total, 100.0 \* correct / total  
history = {**'train\_loss'**: [], **'test\_loss'**: [], **'test\_acc'**: []}  
best\_acc = 0.0  
start\_time = time.time()  
**for** epoch **in** range(1, args.epochs+1):  
 model.train()  
 running\_loss = 0.0  
 **for** i, (inputs, targets) **in** enumerate(trainloader, 1):  
 inputs, targets = inputs.to(device), targets.to(device)  
 optimizer.zero\_grad()  
 outputs = model(inputs)  
 loss = criterion(outputs, targets)  
 loss.backward()  
 optimizer.step()  
 running\_loss += loss.item() \* inputs.size(0)  
 train\_loss = running\_loss / len(trainloader.dataset)  
 test\_loss, test\_acc = evaluate(testloader)  
 scheduler.step()  
 history[**'train\_loss'**].append(train\_loss)  
 history[**'test\_loss'**].append(test\_loss)  
 history[**'test\_acc'**].append(test\_acc)  
 print(**f'Epoch {**epoch**}/{**args.epochs**} TrainLoss={**train\_loss**:.4f} TestLoss={**test\_loss**:.4f} TestAcc={**test\_acc**:.2f}%'**)**if** test\_acc > best\_acc:  
 best\_acc = test\_acc  
 torch.save({**'model\_state'**: model.state\_dict(),  
 **'acc'**: best\_acc,  
 **'epoch'**: epoch}, os.path.join(args.save\_dir, **'best.pth'**))  
total\_time = time.time() - start\_time  
print(**f'Training finished in {**total\_time/60**:.2f} minutes. Best test acc: {**best\_acc**:.2f}%'**)plt.figure(figsize=(10,4))  
plt.subplot(1,2,1)  
plt.plot(range(1, len(history[**'train\_loss'**])+1), history[**'train\_loss'**], label=**'Train Loss'**)  
plt.plot(range(1, len(history[**'test\_loss'**])+1), history[**'test\_loss'**], label=**'Test Loss'**)  
plt.xlabel(**'Epoch'**)  
plt.ylabel(**'Loss'**)  
plt.legend()  
plt.title(**'Loss'**)  
plt.subplot(1,2,2)  
plt.plot(range(1, len(history[**'test\_acc'**])+1), history[**'test\_acc'**], label=**'Test Accuracy'**)  
plt.xlabel(**'Epoch'**)  
plt.ylabel(**'Accuracy (%)'**)  
plt.legend()  
plt.title(**'Test Accuracy'**)  
plt.tight\_layout()  
plt.savefig(os.path.join(args.save\_dir, **'training\_history.png'**), dpi=150)  
print(**f'History plot saved to {**os.path.join(args.save\_dir, **"training\_history.png"**)**}'**)**def** predict\_image(img\_path):img = Image.open(img\_path).convert(**'RGB'**).resize((32,32))  
 x = transform\_test(img).unsqueeze(0).to(device)  
 model.eval()  
 **with** torch.no\_grad():  
 logits = model(x)  
 probs = torch.softmax(logits, dim=1).cpu().numpy()[0]  
 pred = int(np.argmax(probs))  
 **return** img, pred, probs  
**if** args.visualize\_image:  
 img, pred\_idx, probs = predict\_image(args.visualize\_image)  
 plt.figure(figsize=(3,3))  
 plt.imshow(img)  
 plt.axis(**'off'**)  
 plt.title(**f'Pred: {**classes[pred\_idx]**} ({**probs[pred\_idx]\*100**:.1f}%)'**)  
 plt.show()

# Спецификация ввода

Python CIFAR10Classifier <batch\_size>x3x32x32

# Пример

Python CIFAR10Classifier 128x3x32x32

# Спецификация вывода

class = <номер класса от 0 до 9>

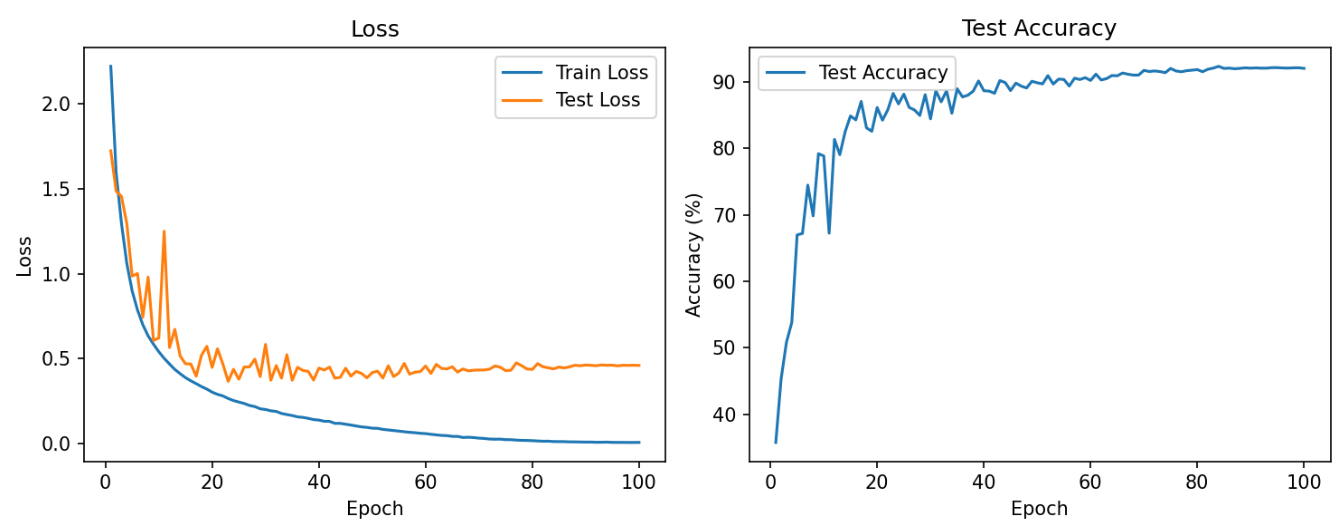
label = <текстовое имя класса>

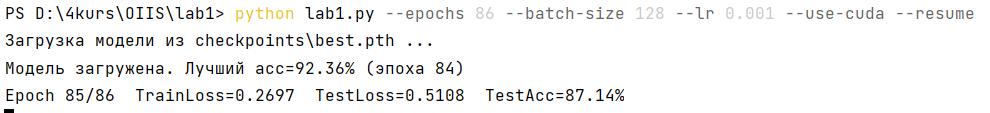
# Пример

class = 3

label = cat

# Рисунки с результатами работы программы

****

****

**Вывод:** научился конструировать нейросетевые классификаторы и выполнять их обучение на известных выборках компьютерного зрения.